OpenKnowledge

FP6-027253

Peer to Peer Coordination Protocol

Adrian Perreau de Pinninck[1], David Dupplaw[2], Spyros Kotoulas[3], Marco Schorlemmer[1], Ronny Siebes[3], and Carles Sierra[1]

[1] Artificial Intelligence Research Institute, IIIA-CSIC, Spain
[2] School of Electronics and Computer Science, University of Southampton, UK
[3] Vrije Universiteit Amsterdam, The Netherlands

Report Version: final
Report Preparation Date:
Classification: deliverable D1.2
Contract Start Date: 1.1.2006 Duration: 36 months
Project Co-ordinator: University of Edinburgh (David Robertson)
Partners: IIIA(CSIC) Barcelona
Vrije Universiteit Amsterdam
University of Edinburgh
KMI, Open University
University of Southampton
University of Trento

DELIVERABLE 1.2
Coordination protocol specification

Adrian Perreau de Pinninck[1], David Dupplaw[2], Spyros Kotoulas[3], Marco Schorlemmer[1], Ronny Siebes[3], and Carles Sierra[1]

[1] Artificial Intelligence Research Institute (IIIA - CSIC)
Barcelona, Spain
adrianp@iiia.csic.es
marco@iiia.csic.es
sierra@iiia.csic.es
[2] School of Electronics and Computer Science
Southampton, UK
dpd@ecs.soton.ac.uk
[3] Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
kot@few.vu.nl
ronny@cs.vu.nl

Abstract. In this report we define the protocols used in the OpenKnowledge platform to coordinate the interacting peers. These protocols are the basics to be implemented by the software installed in each user's machine in order to join the OpenKnowledge community. Once the software is installed the user may search and download other pieces of software (or plug-ins) that will allow him to interact with other users. All these processes are supported by the protocols that we describe in this article.

1 Introduction

This paper explains the underlying protocols for any interaction that goes on within the OpenKnowledge system. OpenKnowledge is a platform based on a P2P paradigm where each OpenKnowledge user accesses the network through an application installed in a computer connected to the internet. In collaboration with their peers, users may bring about many different tasks. Tasks are formalized in OK as *interaction models (IMs)*, which specify how the interaction between many different users is to proceed. In deliverable 1.1 [1] and in [2] the language that will be used to define these interaction models has been defined, namely Ambient LCC. Nonetheless, the OK system is orthogonal to it and is prepared to use other languages to specify interaction models.

The user may interact with other users by playing one of the constitutive roles of an interaction model. *OpenKnowledge Components (OKCs)* are the code plug-ins that allow users to play particular roles in an interaction model. OKCs can be searched and downloaded from the network by the users and installed in their

computers. Once an OKC is installed in the peer, the user can *subscribe* it to play the desired role in the enacted IM. This subscription process lets the team formation service (TFS) know when the OKC is ready to start interacting, that is, when all the roles in the IM are covered by peers. The TFS is executed by some peers of the OpenKnowledge network to steer the team formation process. When the TFS has enough subscriptions to fulfill the obligatory roles in the IM, a team will be formed to start a new interaction. The users forming a team will interact with each other through a coordinator peer in charge of orchestrating the interaction according to the IM. In Figures 1 we see how the different parts of the OpenKnowledge platform are related to each other in a high level module description.

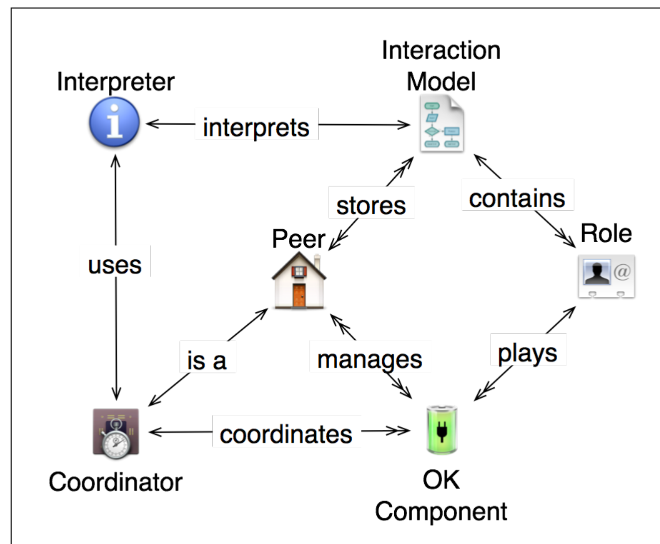

Fig. 1. Open Knowledge Architecture

There are two main protocols involved in these interactions. First, the protocol for finding the players that will make up a team for an interaction (team formation). Second, a protocol to verify that the interaction model is being followed (orchestration). These two protocols have been defined and implemented in the system's first prototype version. We are using an iterative approach to build our software, which means that in future iterations the base protocols are liable to change. We have taken this into account when designing the protocols and we have purposefully left space for future modifications.

In sections 2 and 3 we define in more detail the interaction models and the OpenKnowledge components. In section 4 we explain how the communication layer that serves as a message passing framework has been designed. In section 5 we explain the protocol for team formation and in section 6 we define the base

protocol for interaction enactment. In section 7 we give an example of how the whole system would work with a simple interaction. In section 8 we give insights into the new developments that are planned for future prototypes. Finally, in section 9 we give a summary of the sate of the protocols.

2 Interaction Models

One of the novel aspects of OK is that it aims at raising the profile of the interaction models to the point that they are the network's main currency. This means that every message that is processed on the network always has some context (an interaction model) within which it is being executed. This is very important to guarantee the openness of applications.

OKCs on the system may be seen in an extremely simplified version of its desired full functionality as passive components that provide a data processing functionality, just like a web-service. That is, when called they process data and produce an output. If we were to write the specifications for the inputs and outputs of these components we might end up with a semantic web-service description that would allow components to be executed on some data. This over simplification of the behaviour of an OKC will be removed in future implementations to permit more proactive behaviour.

However, any use of these components to solve large and complex problems has to be defined. This is done by defining the OKC orchestration via the specification of their interaction. In the first implementation it will be a functional interaction that will combined the input and output of the components in a consistent way. In future versions a more complex combination that take into account the autonomy of the OKCs will be permitted. However, even in the current simplified version, from the types of the inputs and outputs of a component, no inference can be made on how to sensibly orchestrate the values that pass between them; the coordination rules of the interaction model will provide such a formalisation. To fully define specific applications in which these components may be activated we thus need to define both the interaction specification and the coordination rules. Together, they create an interaction model by which we can orchestrate components providing services that fulfil the roles within the interaction specification.

Ideally an IM formalisation should be executable; this would allow the components themselves to not only understand the rules of the interaction, but execute the interaction directly from the model itself. Interpreters at the peers provide the means by which incoming interaction model instances can be executed. Section 6 provides further information on how the interaction models are executed in the OpenKnowledge system.

We have based interaction models on message passing protocols with constraints on the generation and reception of certain messages. That is, messages are sent to or received from peers if the constraints on that send or receive messages are satisfied in the current context. In practical terms, this means that constraints are implemented as calls to procedures that in case of success when

acting as constraints in message sending trigger the message operation, and when acting as constraints in message receiving are triggered when the message is received. If the constraints are not satisfied we must backtrack in the execution of the interaction model to find an alternative path in the code.

3 OpenKnowledge Components

OKCs play the active role in an interaction model. Many different OKCs with the ability to play the same role in the same IM can coexist in the system. It will be up to the user to decide which one suits his needs best.

OKC's are made up of the following parts:

- *IM reference* - the identifier of the interaction model that the OKC can play.
- *Role* - the name of the role in the referenced interaction model that the OKC can play.
- *Code* - the algorithms (procedures) that solve the constraints for the message operations that are required by the OKC's role in the referenced interaction model.
- *Description* - (optional) describes how the OKC plays the role in the referenced IM. It gives the user an insight into its workings, helping him decide if he wants to use this OKC or another one.

OKCs are packaged into one file so that they can be moved around the network easily. The OKC file is a zip file which contains the compiled code (optionally the source code too), and an XML file with the IM reference, role, description, and name of the main class.

4 Communication Layer

The communication layer supports the set of basic protocols used by the peers of the OK system to communicate with one another. The main idea is that messages are sent asynchronously between peers using TCP/IP as the transport protocol. This is a simple protocol that is widely spread over the Internet and guarantees that messages are delivered.

We use the pipe concept, through which messages are sent between *EndPoints*. Each EndPoint has an identifier: *EndPointID* that is to be included in messages and that the communication layer uses to route them to the appropriate receiver. There can be many EndPoints within one single peer, therefore messages have to contain information on how to reach the peer and the EndPoint within it. Since we are using TCP/IP as the message sending protocol, we assume each peer has an IP address, and will be listening for messages at a concrete port. Therefore, an EndPointID is made up of the peer's address and port, and a unique identifier inside the peer. Messages contain the sender and receiver's EndPointID and the information being delivered.

Each peer's communication layer is in charge of managing its EndPoints and of sending and receiving messages. When an OKC in the platform expects to

receive messages through the communication layer, it informs the peer which will generate a unique identifier for it, and create the EndPointID with this id, the peer's IP address and the port. The peer maintains a list with the EndPoints it holds in order to forward the messages that it receives to their destination. Each peer will be listening at its port for messages sent to those EndPoints it holds.

When a message is received by the peer, it reads the EndPointID to which it must be delivered (the IP address and port should match the peer's) and extracts the EndPoint's unique identifier. It will look for the EndPoint with the given identifier and will forward the message to it. When the peer is given a message by one of its EndPoints that it has to deliver to another EndPoint, it reads the destination IP and port from the EndPointID and sends the message to the given address. Messages are sent over the network.

5 Team Formation

Team formation is a process achieved through three different subtasks: OKC subscription, coordinator subscription, and team instantiation. Each of these subtasks is achieved by means of a simple protocol:

- OKC subscription - This protocol selects one of his peer's OKC and informs the TFS about the readiness of the OKC in taking part in the execution of its IM. To do so the user first selects the OKC. Then the peer's subscription manager inserts the subscription request into its database. Finally, the peer sends to the Team Formation Service a subscription message containing a reference to the IM, the name of the role to be played, a pointer to the OKC going to play the role, and some extra information about the subscription (i.e., if it is a subscription for a single interaction, or for multiple interactions).
- Coordinator subscription - This protocol starts when a peer configured to play as coordinator is booted. First, the peer creates a running instance of the coordinator. Then, the communication layer creates a new EndPoint and assigns its EndPointID to the coordinator's instance. Finally the peer sends the Team Formation Service a subscription containing the coordinator's EndPointID, and information about the subscription (i.e., the supported IM languages).
- Team instantiation - This protocol is executed by the Team Formation Service every time it receives a new OKC subscription. First the TFS looks for IMs for which there are enough OKC subscriptions to play all of its obligatory roles. Then, for each of those IMs it will find a subscribed coordinator able to parse it. After that, the TFS will send the coordinator the IM and a list of peers for each role in the IM with the name of the OKC subscribed to play. When the coordinator receives this information, it will contact each of the peers in the list to ask them to instantiate the OKC for the new interaction. Each peer's subscription manager will create a new running instance

of the OKC code, the communication layer will create a new EndPointID and assign it to the newly created instance. Finally the peer's subscription manager will inform the coordinator of the EndPointID of the OKC instance playing the role in the interaction model.

6 Orchestration

Orchestration is the process through which an interaction is realized, making sure that the interaction model is respected. There is only one protocol used in this orchestration: the protocol through which the coordinator asks any of the OKCs playing a role in the IM to solve a constraint. As of now, the OKCs are reactive pieces of code that just solve constraints specified in the interaction model, they cannot send messages unless they reply to a previous message sent by the coordinator.

Once the coordinator peer has received a list of EndPointIDs and the interaction model it will be executing, it starts parsing the interaction model code. The parsing of the code will require the intervention of the OKCs to solve the constraints. Therefore, in the parsing process whenever a constraint has to be verified the coordinator will contact the corresponding OKC via its EndPointID to solve it. Communication, and therefore the protocol, are only realized when the coordinator reaches a point in the interaction model where a constraint must be satisfied in order to continue. Constraints are part of the clauses used to model message passing. A message may only be sent if the sending agent solves the constraint. Since it is the OKCs that have the code that can solve interaction model constraints, the coordinator must get in touch with them in order for them to solve the constraint. This is done through what we have called the constraint solving protocol. When the constraint is solved, the coordinator continues executing the interaction model until a new constraint is found or until the interaction model ends.

The constraint solving protocol consists of two parts. The first one is when the coordinator, after finding a constraint in the interaction model it can't solve, sends a message to the OKC which is playing the role that can solve it. The second part occurs when the OKC, after receiving the request to solve a constraint, executes the code that solves this constraint and returns a message to the coordinator with the results.

The constraint solving request is sent by the coordinator. The request is encripted as an XML message sent to the OKC through the communication layer. The coordinator has been given the EndPointIDs of all the interacting OKCs at the beginning of the interaction. The request message contains the name of the constraint to be solved, and a list of the parameters. The parameters can be both input or output values. The request message also contains the interaction state, which contains all the relevant data in the current interaction. The values to be stored in the interaction are defined in the interaction model, and it is the coordinator who is in charge of updating them. When the constraint is received by the peer which is executing the OKC, it reads the name of the constraint to be

solved and looks for the method matching that name in the OKC class. It then executes the method in the class passing the input values and the interaction state as parameters.

When the OKC has executed the method that solves the constraint, the peer gets the returned values and wraps them into a modified interaction state. This state is sent back to the coordinating peer as an XML message. The peer knows the EndPointID of the coordinating peers because it is given in the constraint solving request message. When the coordinator receives the message it merges the new values into the interaction state and continues executing the interaction model.

7 Example

In order for the reader to fully understand the coordination protocol, in this section we will go through a simple *Hello World* example. This example consists of making one peer in the OpenKnowledge system greet another peer. This greeting task is defined through an interaction model which contains two roles (see Figure 2): the greeter and the greetee. The definition of the roles is very simple, the greeter role sends a message to the greetee role which contains the text that is returned by the *greeting* constraint, the greetee role receives a message from the greeter role and its contents are used as a parameter to the *display_greeting* constraint.

```
a(greeter,A) ::
    message(Text) => a(greetee,B) <- greeting(Text)

a(greetee,B) ::
    display_greeting(Text) <- message(Text) <= a(greeter,A)
```

Fig. 2. A simple interaction model

A trivial OKC is defined for each one of the previously defined roles. The one implementing the greeter role, which we will call *writer*, will pop up a window with a text input in which the user will write the greeting it wants delivered. The OKC implementing the greetee role, which we will call *visualizer*, will pop up a window with the text received as a greeting.

Now we will show how a hello world interaction takes place between two peers willing to play its roles. For this we will assume that there are three users with a peer that forms part of the OK network: Anne, John, and Pete. Anne has downloaded the writer OKC and John the visualizer OKC. Pete is one of those users willing to give up its computer power to act as coordinator. The first part of the process would be team formation: Pete would subscribe its peer as a coordinator. John would subscribe its visualizer OKC to play the greetee

role (at this point the TMS still does not have enough subscription to fulfill all the roles of the hello world interaction model). Anne would subscribe its writer OKC to play the greeter role. In each case the user's peer will remember which component has been subscribed and will send the TMS a subscription message that contains the Hello World IM, the name of the role (either greeter or greetee), and the identifier of the OKC. Now the TMS realizes that it has received enough subscriptions to fulfill the two roles of the hello world IM. Therefore, it looks for a coordinator peer, since only Pete's peer is subscribed as coordinator the TMS sends it a message with the Hello World IM, and the list of actors that will take part in the interaction: Anne's writer OKC playing greeter, and John's visualizer OKC playing greetee. At this point the Pete's peer (the coordinator) will send a message to Anne's peer asking it to create a new OKC instance for the interaction. Anne's peer will create an OKC instance which will have and EndPointID. The EndPointID is sent to Pete's coordinator so it can communicate with it. The same will happen for John. At this point there is a coordinator instance running at Pete's peer, that knows that Anne's peer will be playing the greeter role, and John's peer will be playing the greetee role. The coordinator knows both their EndPointIDs.

Now that all the peers are ready, the coordinator peer (John's) starts parsing the interaction model. The role that initiates the interaction is the greeter role, which needs to satisfy the *greeting* constraint. Therefore the coordinator sends a message to the EndPoint of the OKC playing that role (Anne's writer OKC) asking to satisfy the constraint. Anne's peer receives the message and forwards it to the appropriate EndPoint (the writer OKC instance). The OKC pops up a window for Anne to write the greeting text. Anne writes "Hello Buddy!!". This text is returned as satisfying the constraint. The solved constraint is sent as a message to the coordinator. Now the coordinator continues parsing the IM and sees that the greetee role should receive the message with the text sent by the greeter role. And the text is then used as a parameter to the *display_greeting* constraint, that is to be satisfied by the greetee role. The coordinator sends a message with the constraint to be solved, which has as a parameter the text written by Anne: "Hello Buddy!!", to the EndPoint playing the greetee role (John's peer). When the message is received John's visualizer OKC pops up a window with the text used as parameter. Now John has been greeted by Anne, and the Interaction has finished.

8 Future Work

As we already said in the introduction, we are using an iterative approach to develop the platform in which OpenKnowledge will be run. In future iterations we plan to add some improvements that will make the OK architecture more flexible. In this section we explain some of the improvements which we already have in mind.

We have realized a modular design for the OpenKnowledge architecture. This way we have paved the way for future improvements that are not traumatic.

One of these improvements is to allow interaction models to be defined through other languages other than Ambient LCC, which is our prototype language. This will be accomplished by implementing coordinators that interpret the new interaction model definition languages. When a team is formed that plays an interaction defined in a language, the team formation service will search for a coordinator that can interpret them. When a peer subscribes as a coordinator it will have to tell the TFS what languages it can interpret.

We will also implement reputation mechanisms into the team formation protocols. This way OK users will be able to restrict the users with which they wish to interact by adding constraints to the OKC subscription process. Also, before forming a team, the system will inform all the potential teammates about the reputation of other potential teammates, so that the users will be able to decide if they want to join. In order to achieve this we will have to add protocols through which some users can rate OKCs and OK peers in the network, and to fetch these ratings.

We are planning to deal with dynamic teams in future architecture versions. Right now the teams are formed at the beginning of the interaction, and they remain static throughout it. To achieve a system that is more open, we will allow users to join ongoing interactions. This way more types of IMs may be described.

OKCs in this system are fully reactive. We plan to modify the orchestration protocols in order to allow OKCs to be proactive and choose when they want to send messages as part of the interaction. The coordinator peers will still be in charge of enforcing the IM. We also want to decouple OKCs from IMs, by allowing OKCs to execute roles in different IMs. This would mean that the OKC would define its properties and through matching techniques the TFS would recruit the OKCs that have characteristics matching those needed to play each role.

9 Conclusions

In this paper we have shown how we have implemented interaction models between distributed components in the OpenKnowledge project. Through a series of protocols that have been specified in sections 5 and 6, an OpenKnowledge user is able to download code that is able to play a role in an interaction model, and execute this code locally so that it can interact with other computers in the OpenKnowledge network by enacting the specified interaction model.

The following protocols have been specified: OKC subscription to play a role in an interaction model, peer subscription to act as interaction model coordination, team instantiation protocol, and constraint solving protocol. These protocols are to be executed in order to realize an interaction, some of them just once, and others many times.

In the actual implementation of the OpenKnowledge system we have centralized the coordination task, OKCs cannot choose who they interact with or join ongoing interactions, and the communication layer we have used is rather simple. In future versions we plan to relax these assumptions in order to make

the platform more flexible and general purpose. This means that the protocols will have to be more sophisticated, but we have left hooks in our implementation that will allow this.

References

1. Sindhu Joseph, Adrian Perreau de Pinninck, Dave Robertson, Carles Sierra, and Chris Walton. Interaction model definition language. Technical report, Openknowledge consortium, 2006.
2. Sindhu Joseph, Adrian Perreau de Pinninck, Dave Robertson, Carles Sierra, and Chris Walton. Interaction model definition language. In *Agent Organizations: Models and Simulations (AOMS workshop) in IJCAI'07*, 2007.